538824 118 JP2

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００２年１２月　９日

出願番号
Application Number: 特願２００２－３５６９７５

[ST. 10/C]: [ＪＰ２００２－３５６９７５]

出願人
Applicant(s): シャープ株式会社

２００３年１１月　７日

特許庁長官
Commissioner,
Japan Patent Office

今井康夫

出証番号　出証特２００３－３０９２２６９

【書類名】　特許願
【整理番号】　186290
【提出日】　平成14年12月 9日
【あて先】　特許庁長官殿
【国際特許分類】　G06F　7/00
　　G06F 11/00
【発明者】
　【住所又は居所】　大阪府大阪市阿倍野区長池町２２番２２号　シャープ株式会社内
　【氏名】　牧田　廣一
【発明者】
　【住所又は居所】　大阪府大阪市阿倍野区長池町２２番２２号　シャープ株式会社内
　【氏名】　大西　充久
【特許出願人】
　【識別番号】　000005049
　【住所又は居所】　大阪府大阪市阿倍野区長池町２２番２２号
　【氏名又は名称】　シャープ株式会社
【代理人】
　【識別番号】　100062144
　【弁理士】
　【氏名又は名称】　青山　葆
【選任した代理人】
　【識別番号】　100086405
　【弁理士】
　【氏名又は名称】　河宮　治

【選任した代理人】

【識別番号】　100084146

【弁理士】

【氏名又は名称】　山崎　宏

【選任した代理人】

【識別番号】　100122286

【弁理士】

【氏名又は名称】　仲倉　幸典

【手数料の表示】

【予納台帳番号】　013262

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　0208766

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　デバッグ装置、デバッグ方法および記録媒体

【特許請求の範囲】

【請求項１】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段とを備えたことを特徴とするデバッグ装置。

【請求項２】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とするデバッグ装置。

【請求項３】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とするデバッグ装置。

【請求項４】　請求項１乃至３のいずれか一つに記載のデバッグ装置において、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表すことを特徴とするデバッグ装置。

【請求項５】　請求項１乃至３のいずれか一つに記載のデバッグ装置におい

て、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表すことを特徴とするデバッグ装置。

【請求項６】　請求項１乃至３のいずれか一つに記載のデバッグ装置において、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表すことを特徴とするデバッグ装置。

【請求項７】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、

上記デバッグ情報を記憶手段に記憶させることを特徴とするデバッグ方法。

【請求項８】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とするデバッグ方法。

【請求項９】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、

上記デバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とするデバッグ方法。

【請求項１０】　請求項７乃至９のいずれか一つに記載のデバッグ方法を実現させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

この発明は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置およびデバッグ方法に関する。また、そのようなデバッグ方法を実現させるためのプログラムを記録した記録媒体に関する。

【０００２】

なお、並列プログラミング言語により記述されたプログラムを並列プログラムと呼ぶ。

【０００３】

また、逐次処理を記述するためのプログラミング言語を逐次プログラミング言語、それにより記述されたプログラムを逐次プログラムと呼ぶ。

【０００４】

また、デバッグとはプログラムの不具合を特定して修正することを意味する。

【０００５】

【従来の技術】

一般に、並列にいくつかの動作を同時に行う処理を記述する場合、並列プログラミング言語が用いられる。例えば、大規模集積回路（ＬＳＩ）の設計においては、その内部の動作は基本的に並列動作であるため、プログラミング言語としては、逐次処理を記述するプログラミング言語ではなく、並列プログラミング言語が用いられる。この並列プログラミング言語により設計されたＬＳＩの動作を検証するためには、本来、そのＬＳＩの動作が記述された並列プログラムを計算機上で動作させるのが望ましい。

【０００６】

しかし、ＬＳＩの設計者が設計のために使用している計算機は、逐次プログラムを実行するものであることが多い。そのような逐次プログラムを実行する計算機を使って、並列プログラムを直接実行することはできない。

【０００７】

そこで、図１に示すように、並列プログラム１を逐次化装置２によって自動的

に逐次プログラム４に変換し、得られた逐次プログラム４をその計算機上で実行する方法が広く行われている。逐次プログラム４を実行した結果、元の並列プログラム１にバグ（不具合）があることが分かった場合、設計者は逐次プログラム４をデバッガ６上で動作させて不具合個所を特定し、その後、元の並列プログラム１の対応する個所を修正する。

【０００８】

【発明が解決しようとする課題】

ところで、並列プログラム１を逐次化装置２によって自動的に逐次プログラム４に変換する際に、並列プログラム１での一つの行が、逐次プログラム４では複数の行からなる処理に書換えられる場合がある。また、変数名についても、並列プログラム１での或る変数名が、逐次プログラム４では別の変数名に書き換えられる場合がある。

【０００９】

しかし、上述の方法では、並列プログラム１から逐次プログラム４への変換時には、並列プログラム１での或る行が逐次プログラム４ではどの行に書換えられたかや、並列プログラム１での或る変数名が逐次プログラム４ではどの変数名に書換えられたかの情報が残らない。

【００１０】

そのため、デバッグを行う作業者は、並列プログラム１での行と逐次プログラム４での行との対応を自身で取りながら作業を行わなければならない。また、並列プログラムでの変数名と逐次プログラムでの変数名との対応も自身で取りながら作業を行わなければならない。このため、並列プログラムを逐次プログラムに変換してデバッグを行う作業は、非常に効率が悪くなり、時間がかかるという問題がある。

【００１１】

また、図２に示すように、並列プログラム１を一旦逐次プログラム４に変換し、逐次プログラム４に不具合があった場合は、元の並列プログラム１ではなく逐次プログラム４を修正し、修正後の逐次プログラムを並列化装置９によって並列プログラム１０に変換する方法も知られている（例えば、特許文献１参照。）。

【００１２】

しかし、この方法によっても、作業者は、並列プログラム１におけるのとは異なる行や変数名を含む逐次プログラム４をデバッグしなければならない。このためデバッグ作業の効率が悪くなる。また、この方法によれば、逐次プログラム４をデバッグした後に並列プログラム１０に変換するため、得られた並列プログラム１０が、元の並列プログラム１の作成者が意図しない構造に変更されてしまう可能性がある。そのため、プログラムの分り易さが低下し、後で、並列プログラム１０を使用または変更しようとする際の作業効率が下がる。

【００１３】

【特許文献１】

特開平８－１６４２９号公報

【００１４】

そこで、この発明の課題は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、作業者がデバッグ作業を効率良く行うことができるものを提供することにある。

【００１５】

また、この発明の課題は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、作業者がデバッグ作業を効率良く行うことができるものを提供することにある。

【００１６】

また、この発明の課題は、そのようなデバッグ方法を実現させるためのプログラムを記録した記録媒体を提供することにある。

【００１７】

【課題を解決するための手段】

上記課題を解決するため、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段とを備えたことを特徴とする。

【００１８】

この発明のデバッグ装置では、逐次化手段は、並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、記憶手段が上記デバッグ情報を記憶する。したがって、デバッグを行う作業者は、上記記憶手段の記憶内容、つまり上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００１９】

また、別の面では、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とする。

【００２０】

この発明のデバッグ装置では、記憶手段は、デバッグ対象となった並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶している。変換手段は、上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００２１】

また、別の面では、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とする。

【００２２】

この発明のデバッグ装置では、逐次化手段は、並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、記憶手段が上記デバッグ情報を記憶する。変換手段は、上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００２３】

上記並列プログラムでの情報を表示する一方、その情報に対応した上記逐次プログラムでの情報を隠すのが望ましい。そのようにした場合、作業者は上記変換手段を介してあたかも上記並列プログラムに対して直接デバッグを行っているかのような感覚でデバッグ作業を行うことができる。例えば、作業者は元の並列プログラム内の行番号や変数名を使って不具合箇所を特定することができる。そして、元の並列プログラム内の行番号や変数名を使ってその不具合を修正して解消することができる。

【００２４】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表すことを特徴とする。

【００２５】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で行番号の対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で対応する行番号同士を相

互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【００２６】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表すことを特徴とする。

【００２７】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で変数名の対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で対応する変数名同士を相互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【００２８】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表すことを特徴とする。

【００２９】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で行番号の対応と変数名の対応とを容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で対応する行番号同士、変数名同士を相互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【００３０】

この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、

上記デバッグ情報を記憶手段に記憶させることを特徴とする。

【００３１】

この発明のデバッグ方法では、逐次化手段によって並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、上記デバッグ情報を記憶手段に記憶させる。したがって、デバッグを行う作業者は、上記記憶手段の記憶内容、つまり上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００３２】

また、別の面では、この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とする。

【００３３】

この発明のデバッグ方法では、デバッグ対象となった並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させておく。そして、上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００３４】

また、別の面では、この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、

上記デバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とする。

【００３５】

この発明のデバッグ方法では、逐次化手段によって並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、上記デバッグ情報を記憶手段に記憶させる。上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００３６】

この発明の記録媒体は、上述のいずれか一つのデバッグ方法を実現させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体である。

【００３７】

この発明の記録媒体に記録されたプログラムをコンピュータ読み取りして実行すれば、上述のデバッグ方法を実現することができる。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００３８】

【発明の実施の形態】

以下、この発明を図示の実施の形態により詳細に説明する。

【００３９】

図３は、本発明の一実施形態のデバッグ装置の構成を示している。

【００４０】

このデバッグ装置は、図中の並列プログラム３１を逐次プログラム３４に変換してデバッグを行うために、逐次化手段としての逐次化部３２と、記憶手段とし

てのデータベース３５と、逐次プログラムをデバッグするツールとしてのデバッガ３６と、変換手段としてのデバッガ・インタフェース３７とを備えている。

【００４１】

逐次化部３２は、逐次化処理を行って並列プログラム３１を逐次プログラム３４に変換する。なお、逐次化処理としては、公知の方法を用いることができる。

【００４２】

逐次化部３２の内部にはデバッグ情報生成部３３が含まれている。デバッグ情報生成部３３は、逐次化処理と同時に、並列プログラム３１と逐次プログラム３４との対応関係を表すデバッグ情報を生成する。このデバッグ情報はデータベース３５に格納される。

【００４３】

デバッガ３６は、作業者が入力したコマンドをデバッガ・インタフェース３７を介して受け取り、受け取ったコマンドに応じて逐次プログラム３４をデバッグする。

【００４４】

デバッガ・インタフェース３７は、デバッグ作業者３８からコマンドを受け取り、データベース３５を参照して、必要に応じて変換したコマンドをデバッガ３６に対して発行する。また、デバッガ３６からの出力を受け取り、データベース３５を参照して、必要に応じて変換した出力をデバッグ作業者３８に向けて出力する。

【００４５】

なお、このデバッグ装置は、上述の構成要素以外に、デバッグ対象となったプログラム表示し得る表示装置や、デバッグ作業の出力を作業者に提供し得る出力装置を備えている。

【００４６】

次に、上記の様に構成されたデバッグ装置によってデバッグを行う処理を説明する。

【００４７】

ｉ）　まず、並列プログラム３１として図４に示すような内容を持つものが、

逐次化部３２に入力される。

【００４８】

ここで、図４中の左欄に示す行番号（１０，１１，…，２９）４５は、説明のために付加したものであり、並列プログラム３１には含まれない。

【００４９】

この並列プログラム３１は、Ｃ言語を拡張した文法をもっており、キーワード“ｐａｒ”の後の中括弧 {| で囲まれた範囲内に、並列に実行する処理が記述されている。この例では、点線で囲まれた部分の処理４１、４２は並列に実行される処理である。これらの処理４１、４２は、互いに独立した処理であるため、互いに同じ名前の変数ｉや変数ｋなどを含んでいても構わない。理解の容易のために、処理４１に含まれた変数ｉに符号４３、処理４２に含まれた変数ｉに符号４４を付している。

【００５０】

ii)　上記並列プログラム３１を逐次化部３２がＣ言語を用いて逐次化して、逐次プログラム３４として図５乃至図６に示すような内容を持つものが出力される。

【００５１】

なお、図５は前半部分、図６は後半部分に相当する。図５、図６中の左欄に示す行番号（１００，１０１，…，１５５）５５は、説明のために付加したものであり、逐次プログラム３４には含まれない。

【００５２】

この逐次プログラム３４は、点線で囲まれた部分の処理５１、５２が、図４中の処理４１、４２にそれぞれ対応しており、逐次動作するようになっている。この逐次プログラム３４での変数ｉ＿０（符号５３で示す）、変数ｉ＿１’（符号５４で示す）は、それぞれ並列プログラム３１中の処理４１での変数ｉ（符号４３で示す）、処理４２での変数ｉ（符号４４で示す）に対応している。このように、処理４１、４２に含まれた同じ変数名の後ろにそれぞれ＿０、＿１というようなスコープ番号を付して、それらを区別している。この理由は、逐次プログラム（の一つの処理ブロック内）では、同じ変数名は同じ実体を表すものとして

扱われ、同じ変数名であって実体が異なるものが存在することは許されないからである。

【００５３】

この逐次プログラム３４に含まれた処理５１、処理５２は、図５中の点線で囲まれた部分以外の処理にしたがって、疑似的に並列に実行される。

【００５４】

iii)　逐次化部３２が並列プログラム３１を逐次プログラム３４へ変換すると同時に、逐次化部３２内部のデバッグ情報生成部３３が、並列プログラム３１と逐次プログラム３４との対応関係を表すデバッグ情報を生成する。生成されたデバッグ情報はデータベース３５に格納される。

【００５５】

この例では、デバッグ情報は、図７に示す行番号対応情報６１と、図８に示す変数名対応情報６２と、図９に示すスコープ情報６３とを含んでいる。

【００５６】

図７に示す行番号対応情報６１は、並列プログラム３１で或る処理を行う行番号（右欄６５に並べて示す）と、逐次プログラム３４でその処理と同じ処理を行う行番号（左欄６４に並べて示す）との対応関係を表している。例えば、並列プログラム（“ｐｒｏｇ＿ｐａｒ．ｃ”）の１５行目は、逐次プログラム（“ｐｒｏｇ＿ｓｅｑ．ｃ”）の１１７行目に対応することを示す。この行番号対応情報６１は、逐次化部３２が並列プログラム３１内の行を逐次プログラム３４の行に変換する際に、デバッグ情報生成部３３が、元の行の並列プログラム３１内での行番号と変換先の逐次プログラム３４内での行番号とを関連付けてデータベース３５に登録することにより、生成される。

【００５７】

図８に示す変数名対応情報６２は、並列プログラム３１内での或るデータを保持する変数名（右欄６７に並べて示す）と、逐次プログラム３４内でのそのデータと同じデータを保持する変数名（左欄６６に並べて示す）との対応関係を表している。例えば、並列プログラム３１での変数ｉは、逐次プログラム３４での変数ｉ＿０と変数ｉ＿１に対応することを示す。また、並列プログラム３１での変

数ｋは、逐次プログラム３４での変数ｋ＿０と変数ｋ＿１に対応することを示す。この変数名対応情報６２は、逐次化部３２が並列プログラム３１での変数名を逐次プログラム３４での変数名に変換する際に、デバッグ情報生成部３３が、並列プログラム３１での変数名と逐次プログラム３４での変数名とを関連付けてデータベース３５に登録することにより、生成される。

【００５８】

図９に示すスコープ情報６３は、並列プログラム３１内での変数の有効範囲（始行と終行との二つの行番号で指定される。左欄６８に並べて示す）とスコープ番号（右欄６９に並べて示す）との対応関係を表している。スコープ番号は、並列プログラム３１での変数の有効範囲をそれぞれ区別するために、逐次化処理に際して導入される番号である。例えば、並列プログラム（“ｐｒｏｇ＿ｐａｒ．ｃ”）の１３行目から１８行目までの範囲は、スコープ番号０に対応することを示す。また、並列プログラム（“ｐｒｏｇ＿ｐａｒ．ｃ”）の２１行目から２６行目までの範囲は、スコープ番号１に対応することを示す。このスコープ情報６３は、逐次化部３２が入力の並列プログラム３１の構造を解析する際に、デバッグ情報生成部３３が、並列プログラム３１内の変数の有効範囲（二つの行番号）とその範囲に対応したスコープ番号とを関連付けてデータベース３５に登録することによって、生成される。

【００５９】

iv)　上述のようなデバッグ情報がデータベース３５に格納された後、作業者がデバッガ・インタフェース３７を介してデバッガ３６を動作させて、デバッグを行う。

【００６０】

図１０は、デバッグ作業者がコマンドを発行したときの処理の流れを示している。

【００６１】

まず、作業者３８は、デバッガ・インタフェース３７に、デバッグを行うためのコマンドを発行する。

【００６２】

デバッガ・インタフェース３７は、データベース３５内の行番号対応情報６１（図７に示す）を参照して、そのコマンドに並列プログラム３１での行番号が含まれているか否かを調べる（ステップＳ１）。

【００６３】

そのコマンドに並列プログラム３１での行番号が含まれている場合はステップＳ２へ進む一方、含まれていない場合はステップＳ３へ進む。

【００６４】

ステップＳ２では、データベース３５内の行番号対応情報６１を参照した結果に基づいて、並列プログラム３１での行番号を逐次プログラム３４での対応する行番号に変換する。そして、ステップＳ３へ進む。

【００６５】

ステップＳ３では、データベース３５内の変数名対応情報６２（図８に示す）を参照して、そのコマンドに並列プログラム３１での変数名が含まれているか否かを調べる。

【００６６】

そのコマンドに並列プログラム３１での変数名が含まれている場合はステップＳ４へ進む一方、含まれていない場合はステップＳ９へ進む。

【００６７】

ステップＳ４では、データベース３５内の変数名対応情報６２を参照した結果に基づいて、並列プログラム３１での変数名が逐次プログラム３４での複数の変数名に対応しているか否かを調べる。

【００６８】

並列プログラム３１での変数名が逐次プログラム３４での複数の変数名に対応している場合はステップＳ５－Ｓ７へ進む一方、複数の変数名に対応していない場合、つまり１つの変数名のみに対応している場合はステップＳ８へ進む。

【００６９】

ステップＳ５－Ｓ７では、並列プログラム３１での変数名が逐次プログラム３４での複数の変数名に対応している場合であるから、複数の変数名をそれぞれ区別するためにスコープ番号を利用する。具体的には、まずステップＳ５では、現

在デバッグを行っているプログラム上の位置（これを「デバッグ位置」と呼ぶ。）を調べる。続いて、ステップＳ６では、データベース３５内のスコープ情報６３（図９に示す）を参照して、ステップＳ５で取得した現在のデバッグ位置を含む範囲に対応するスコープ番号を得る。続いて、ステップＳ７では、ステップＳ６で得たスコープ番号を利用して、並列プログラム３１での変数名を逐次プログラム３４での対応する変数名に変換する。例えば、ステップＳ６で得たスコープ番号が０であれば、並列プログラム３１での変数ｉは逐次プログラム３４での変数ｉ＿０に変換される。また、ステップＳ６で得たスコープ番号が１であれば、並列プログラム３１での変数ｉは逐次プログラム３４での変数ｉ＿１に変換される。この後、ステップＳ９へ進む。

【００７０】

ステップＳ８では、並列プログラム３１での変数名が逐次プログラム３４での１つの変数名のみに対応している場合であるから、スコープ番号を用いる必要はない。そこで、単に、データベース３５内の変数名対応情報６２を参照した結果に基づいて、並列プログラム３１での変数名を逐次プログラム３４での対応する変数名に変換する。そして、ステップ９へ進む。

【００７１】

ステップＳ９では、上述のステップＳ１－Ｓ８の処理を経て変換されたコマンドをデバッガ３６に対して発行する。ただし、コマンドが行番号と変数名とのいずれも含まない場合（ステップＳ１，Ｓ３でいずれもＮＯの場合）は、コマンドは変換されずにそのまま発行される。

【００７２】

図１１は、デバッガ３６からの出力に対する処理の流れを示している。

【００７３】

デバッガ３６からの出力を受けると、まず、デバッガ・インタフェース３７は、データベース３５内の行番号対応情報６１（図７に示す）を参照して、その出力に逐次プログラム３４での行番号が含まれているか否かを調べる（ステップＳ１１）。

【００７４】

その出力に逐次プログラム３４での行番号が含まれている場合はステップＳ１２へ進む一方、含まれていない場合はステップＳ１３へ進む。

【００７５】

ステップＳ１２では、データベース３５内の行番号対応情報６１を参照した結果に基づいて、逐次プログラム３４での行番号を並列プログラム３１での対応する行番号に変換する。そして、ステップＳ１３へ進む。

【００７６】

ステップＳ１３では、データベース３５内の変数名対応情報６２（図８に示す）を参照して、その出力に逐次プログラム３４での変数名が含まれているか否かを調べる。

【００７７】

その出力に逐次プログラム３４での変数名が含まれている場合はステップＳ１４へ進む一方、含まれていない場合はステップＳ１５へ進む。

【００７８】

ステップＳ１４では、データベース３５内の変数名対応情報６２を参照した結果に基づいて、逐次プログラム３４での変数名を並列プログラム３１での対応する変数名に変換する。そして、ステップ１５へ進む。

【００７９】

ステップＳ１５では、上述のステップＳ１１－Ｓ１４の処理を経て変換された出力を作業者に対して提供する。ただし、出力が行番号と変数名とのいずれも含まない場合（ステップＳ１１，Ｓ１３でいずれもＮＯの場合）は、出力は変換されずにそのまま提供される。

【００８０】

上述のようにしてデバッグを行うようにした場合、作業者は、並列プログラム３１と逐次プログラム３４との間で対応する行番号同士、変数名同士を自身で相互に変換する必要がない。

【００８１】

さらに、表示装置で並列プログラム３１での情報を表示する一方、その情報に対応した逐次プログラム３４での情報を隠すようにした場合、作業者はデバッガ

・インターフェース３７を介してあたかも並列プログラム３１に対して直接デバッグを行っているかのような感覚でデバッグ作業を行うことができる。例えば、作業者は元の並列プログラム３１内の行番号や変数名を使って不具合箇所を特定することができる。そして、元の並列プログラム３１内の行番号や変数名を使ってその不具合を修正して解消することができる。

【００８２】

したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【００８３】

なお、このデバッグ装置は、逐次プログラムを実行可能な一般的な電子計算機上で、上述のデバッグを行う処理を実行するためのプログラム（ソフトウェア）を用いて構成され得る。そのプログラムは、電子計算機が備える外部記憶装置（固定記憶装置）に記憶されていても良い。また、記録媒体（コンパクトディスク等）に書き込まれたそのようなプログラムを、読み取り可能なドライブで読み取るようになっていても良い。

【００８４】

【発明の効果】

以上より明らかなように、この発明のデバッグ装置およびデバッグ方法によれば、作業者がデバッグ作業を効率良く行うことができる。

【００８５】

また、この発明の記録媒体によれば、そのようなデバッグ方法を実現させることができる。

【図面の簡単な説明】

【図１】　並列プログラムを逐次プログラムに変換しでデバッグを行う従来の方法を説明する図である。

【図２】　元の並列プログラムではなく逐次プログラムを修正し、修正後の逐次プログラムを並列プログラムに変換する従来の方法を説明する図である。

【図３】　本発明の一実施形態のデバッグ装置の構成を示す図である。

【図４】　並列プログラムの例を示す図である。

【図５】　図４の並列プログラムを変換して得られた逐次プログラム（前半

部分）を示す図である。

【図６】　図４の並列プログラムを変換して得られた逐次プログラム（後半部分）を示す図である。

【図７】　データベースに格納されたデバッグ情報としての行番号対応情報を例示する図である。

【図８】　データベースに格納されたデバッグ情報としての変数名対応情報を例示する図である。

【図９】　データベースに格納されたデバッグ情報としてのスコープ情報例示する図である。

【図１０】　デバッグを行う処理（コマンド発行時）のフローを示す図である。

【図１１】　デバッグを行う処理（結果出力時）のフローを示す図である。

【符号の説明】

３１　並列プログラム

３２　逐次化部

３３　デバッグ情報生成部

３４　逐次プログラム

３５　データベース

３６　デバッガ

３７　デバッガ・インタフェース

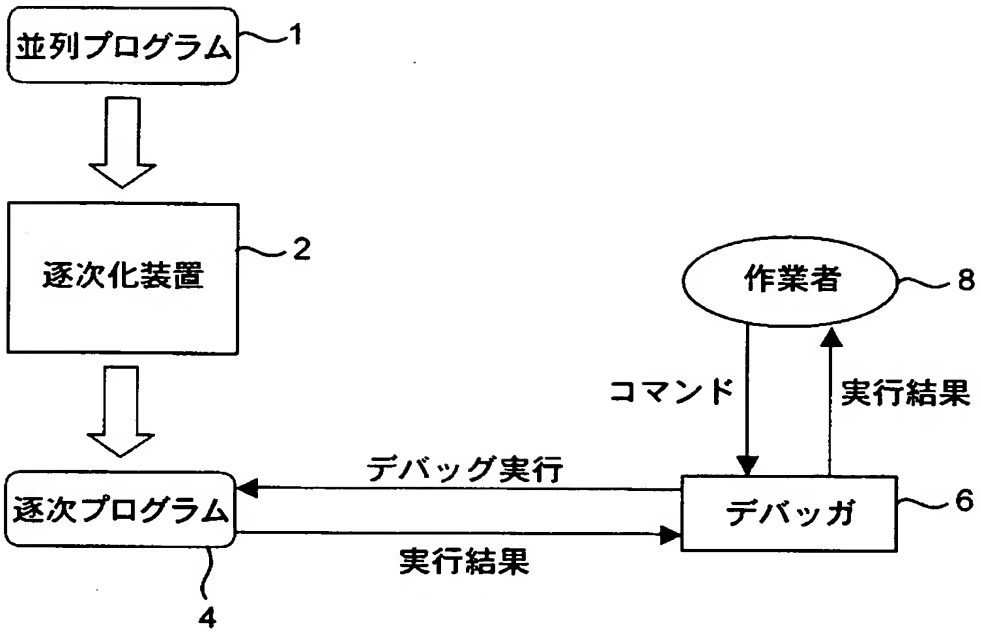
６１　行番号対応情報

６２　変数名対応情報

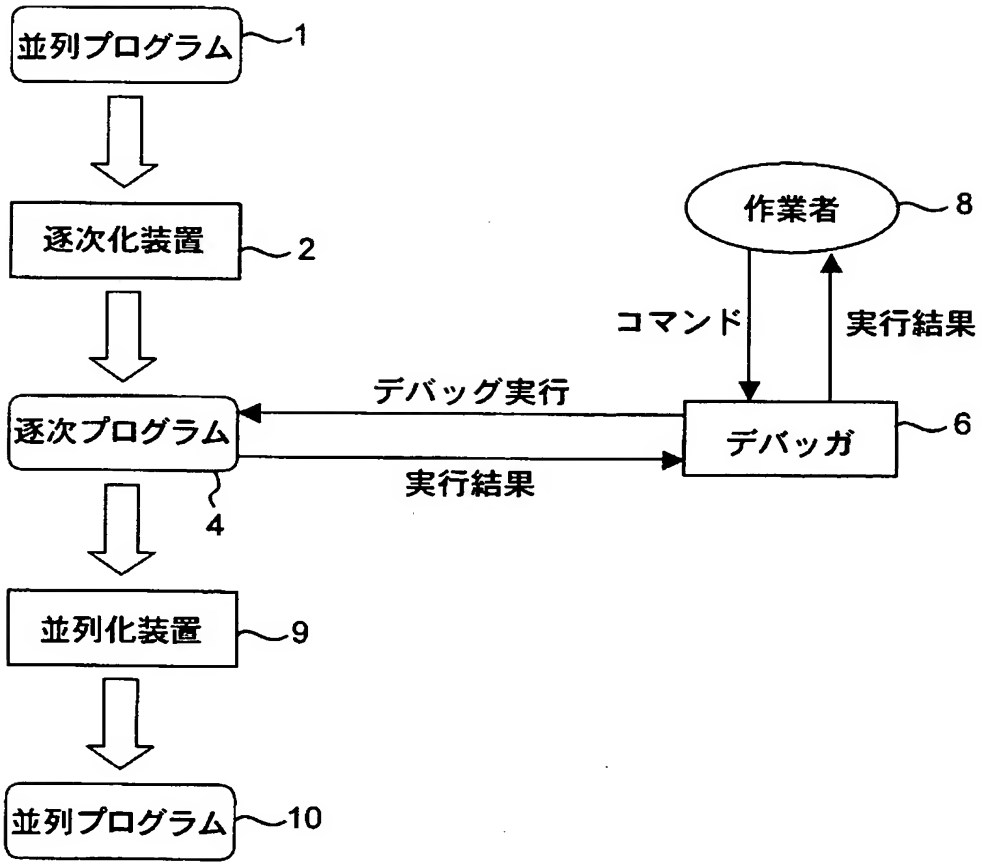
６３　スコープ情報

【書類名】　図面

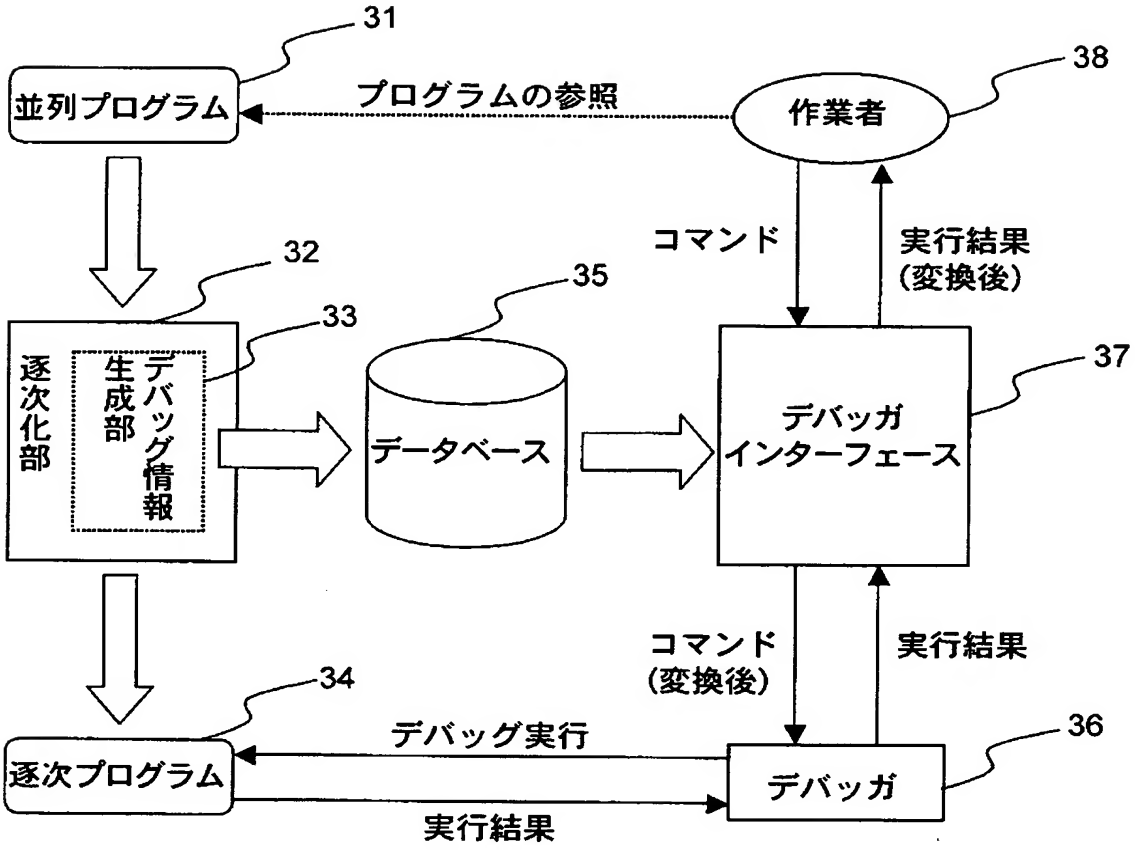
【図１】



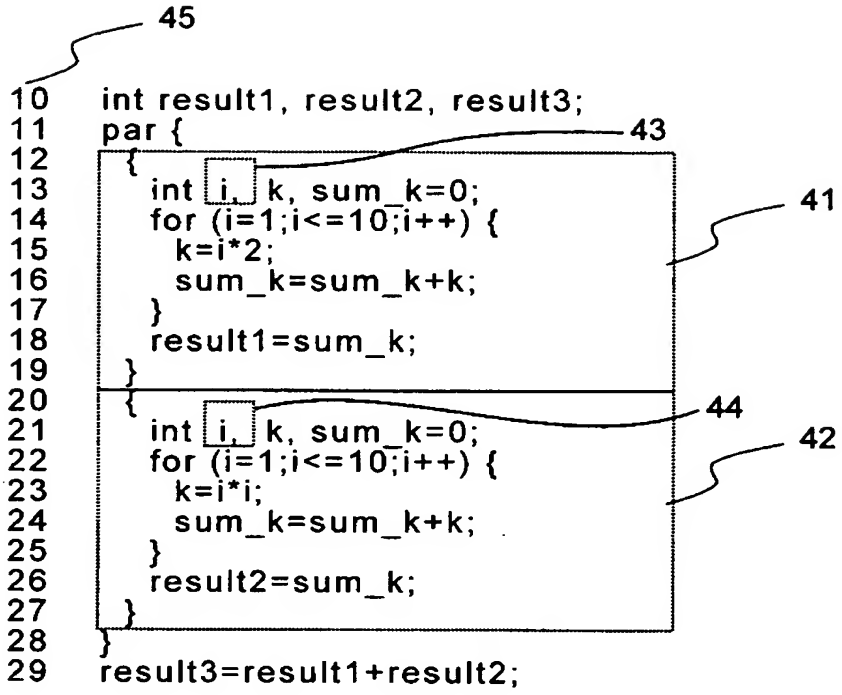
【図２】


並列プログラム
1
逐次化装置
2
逐次プログラム
4
並列化装置
9
並列プログラム
10
作業者
8
コマンド
実行結果
デバッグ実行
実行結果
デバッガ
6

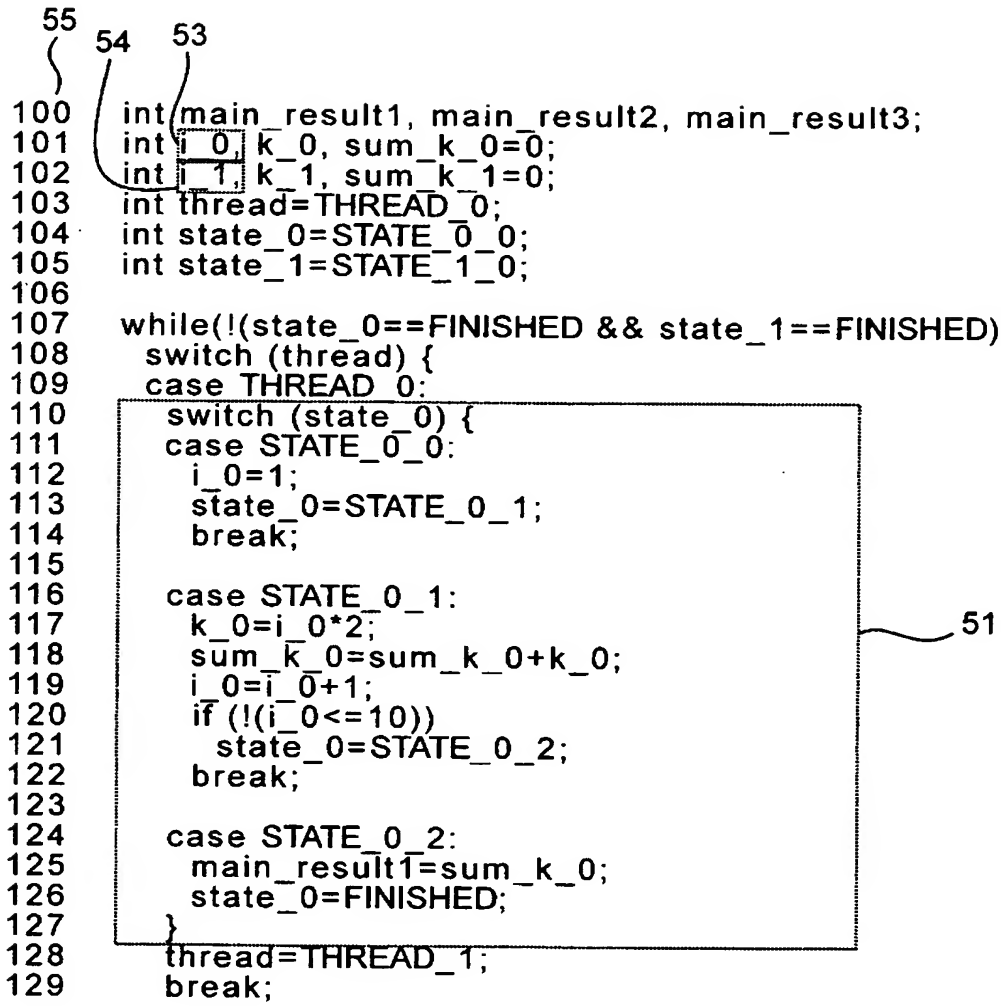
【図３】


31
並列プログラム
プログラムの参照
38
作業者
コマンド
実行結果
(変換後)
32
33
逐次化部
デバッグ情報
生成部
35
データベース
デバッガ
インターフェース
37
コマンド
(変換後)
実行結果
34
デバッグ実行
逐次プログラム
36
デバッガ
実行結果

【図４】


45
10 int result1, result2, result3;
11 par {
12 {
13 int i, k, sum_k=0;
14 for (i=1;i<=10;i++) {
15 k=i*2;
16 sum_k=sum_k+k;
17 }
18 result1=sum_k;
19 }
20 {
21 int i, k, sum_k=0;
22 for (i=1;i<=10;i++) {
23 k=i*i;
24 sum_k=sum_k+k;
25 }
26 result2=sum_k;
27 }
28 }
29 result3=result1+result2;
43
41
44
42

【図５】

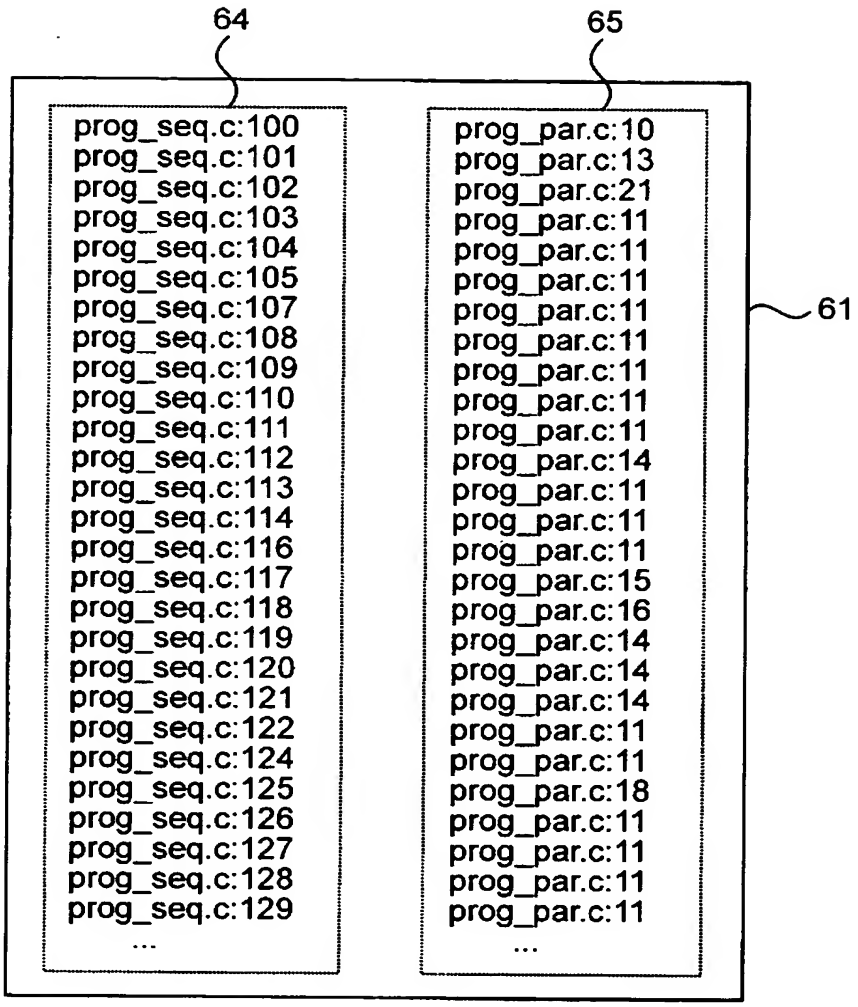

55
54
53
100 int main_result1, main_result2, main_result3;
101 int i_0, k_0, sum_k_0=0;
102 int i_1, k_1, sum_k_1=0;
103 int thread=THREAD_0;
104 int state_0=STATE_0_0;
105 int state_1=STATE_1_0;
106
107 while(!(state_0==FINISHED && state_1==FINISHED)) {
108 switch (thread) {
109 case THREAD_0:
110 switch (state_0) {
111 case STATE_0_0:
112 i_0=1;
113 state_0=STATE_0_1;
114 break;
115
116 case STATE_0_1:
117 k_0=i_0*2;
118 sum_k_0=sum_k_0+k_0;
119 i_0=i_0+1;
120 if (!(i_0<=10))
121 state_0=STATE_0_2;
122 break;
123
124 case STATE_0_2:
125 main_result1=sum_k_0;
126 state_0=FINISHED;
127 }
128 thread=THREAD_1;
129 break;
51

【図６】

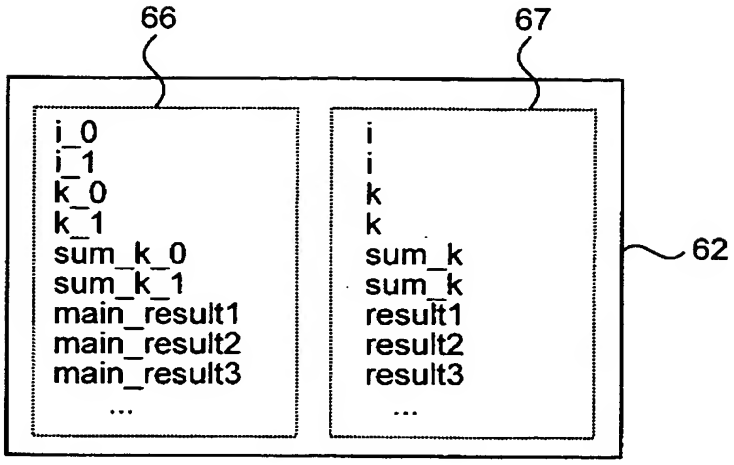
```
130
131     case THREAD_1:
132       switch (state_1) {
133       case STATE_1_0:
134         i_1=1;
135         state_1=STATE_1_1;
136         break;
137
138       case STATE_1_1:
139         k_1=i_1*i_1;
140         sum_k_1=sum_k_1+k_1;
141         i_1=i_1+1;
142         if (!(i_1<=10))
143           state_1=STATE_1_2;
144         break;
145
146       case STATE_1_2:
147         main_result2=sum_k_1;
148         state_1=FINISHED;
149       }
150       thread=THREAD_0;
151       break;
152     }
153   }
154
155 main_result3=main_result1+main_result2;
```

52

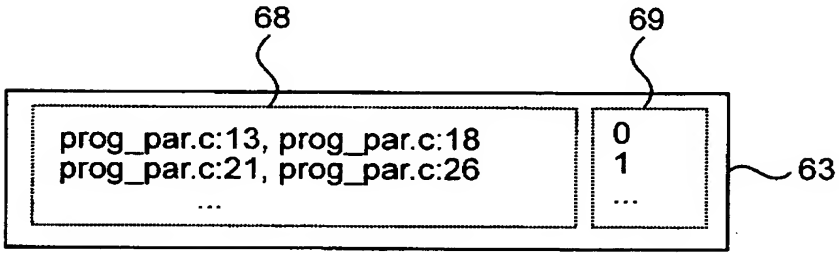
【図７】


64
65
prog_seq.c:100
prog_seq.c:101
prog_seq.c:102
prog_seq.c:103
prog_seq.c:104
prog_seq.c:105
prog_seq.c:107
prog_seq.c:108
prog_seq.c:109
prog_seq.c:110
prog_seq.c:111
prog_seq.c:112
prog_seq.c:113
prog_seq.c:114
prog_seq.c:116
prog_seq.c:117
prog_seq.c:118
prog_seq.c:119
prog_seq.c:120
prog_seq.c:121
prog_seq.c:122
prog_seq.c:124
prog_seq.c:125
prog_seq.c:126
prog_seq.c:127
prog_seq.c:128
prog_seq.c:129
...
prog_par.c:10
prog_par.c:13
prog_par.c:21
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:14
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:15
prog_par.c:16
prog_par.c:14
prog_par.c:14
prog_par.c:14
prog_par.c:11
prog_par.c:11
prog_par.c:18
prog_par.c:11
prog_par.c:11
prog_par.c:11
prog_par.c:11
...
61

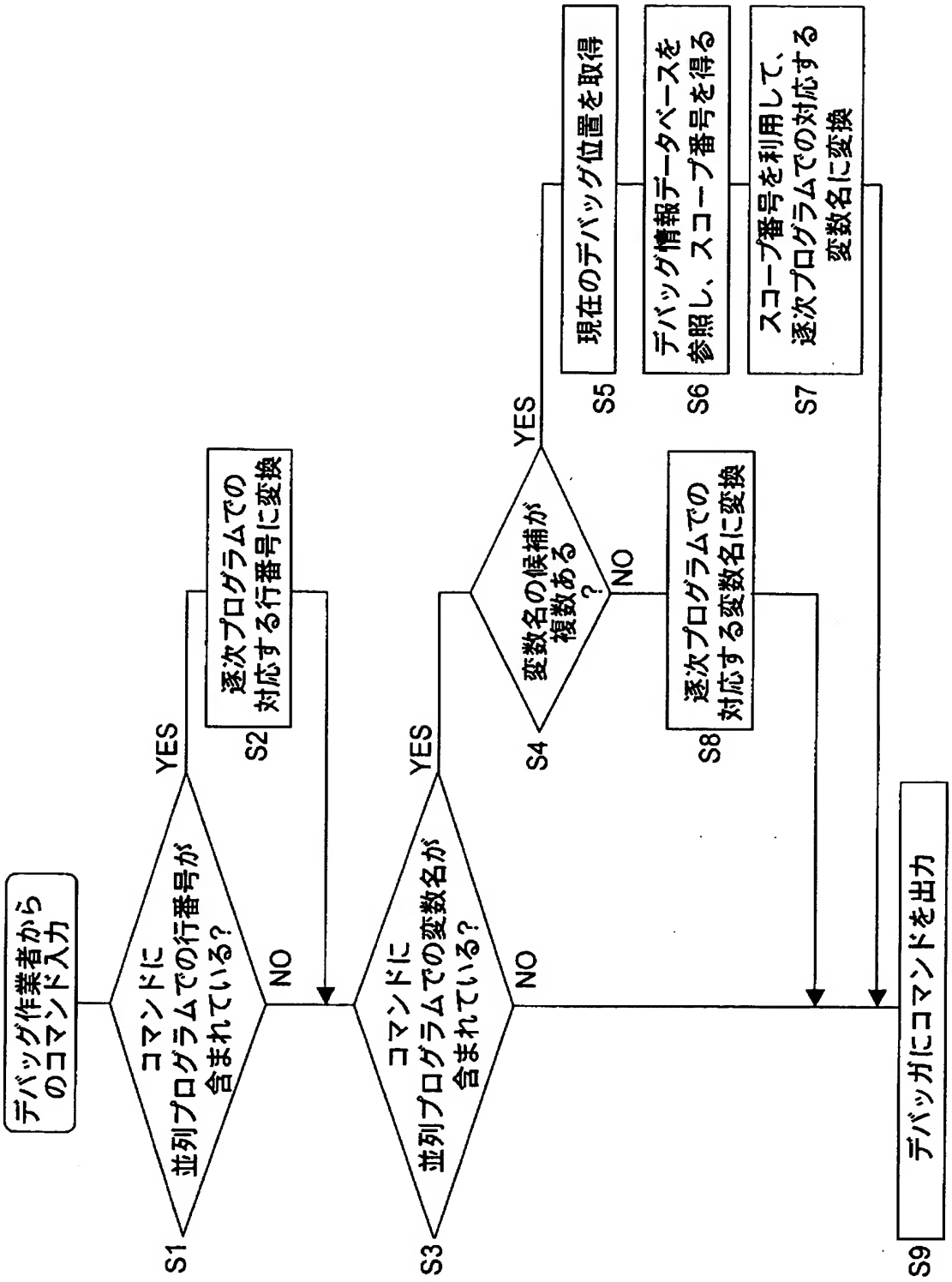
【図８】


66
67
i_0
i_1
k_0
k_1
sum_k_0
sum_k_1
main_result1
main_result2
main_result3
...
i
i
k
k
sum_k
sum_k
result1
result2
result3
...
62

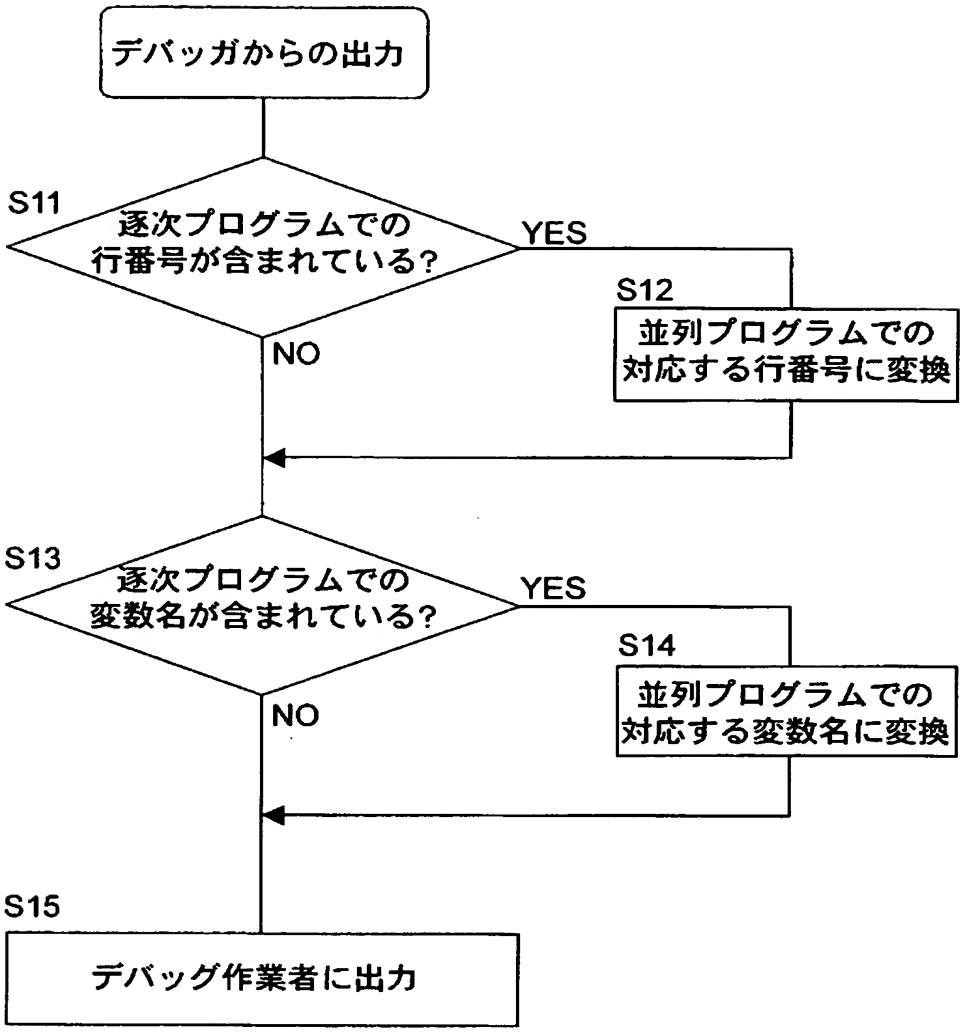
【図９】


68
69
prog_par.c:13, prog_par.c:18
prog_par.c:21, prog_par.c:26
...
0
1
...
63

【図１０】


デバッグ作業者からのコマンド入力
コマンドに並列プログラムでの行番号が含まれている？
S1
YES
NO
逐次プログラムでの対応する行番号に変換
S2
コマンドに並列プログラムでの変数名が含まれている？
S3
YES
NO
変数名の候補が複数ある？
S4
YES
NO
現在のデバッグ位置を取得
S5
デバッグ情報データベースを参照し、スコープ番号を得る
S6
逐次プログラムでの対応する変数名に変換
S8
スコープ番号を利用して、逐次プログラムでの対応する変数名に変換
S7
デバッガにコマンドを出力
S9

【図１１】


デバッガからの出力
S11
逐次プログラムでの
行番号が含まれている?
YES
S12
並列プログラムでの
対応する行番号に変換
NO
S13
逐次プログラムでの
変数名が含まれている?
YES
S14
並列プログラムでの
対応する変数名に変換
NO
S15
デバッグ作業者に出力

【書類名】　要約書

【要約】

【課題】　並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、作業者がデバッグ作業を効率良く行うことができるものを提供すること。

【解決手段】　並列プログラム３１を逐次プログラム３４に変換するとともに、並列プログラム３１と逐次プログラム３４との対応関係を表すデバッグ情報を生成する逐次化手段３２を備える。そのデバッグ情報を記憶する記憶手段３５を備える。そのデバッグ情報に基づいて、並列プログラム３１と逐次プログラム３４との間で対応する情報同士を相互に変換する変換手段３７を備える。

【選択図】　図３

特願２００２－３５６９７５

出　願　人　履　歴　情　報

識別番号　　［０００００５０４９］

１．変更年月日　　１９９０年　８月２９日
［変更理由］　　新規登録
住　所　　大阪府大阪市阿倍野区長池町２２番２２号
氏　名　　シャープ株式会社